Chapter 8: Generic Programming

# 8.1 Why Generic Programming?

-**Generic Programming**: writing code that can be reused for objects of many different types. Example: ArrayList

## 8.1.1 The advantage of Type Parameter

-ArrayList class has a **type parameter** that **indicates** the **element type**: 

->Make programs easier to read and safer.

## 8.1.2 Who wants to be a Generic Programmer?

-**Wildcard type**: **abstract** but allow a library builder to make methods as flexible as possible.

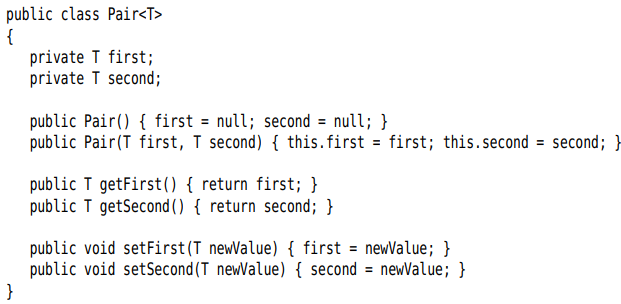
-**Generic programming** has **3 skill levels**: use generic classes without thinking how and why they work – Learn enough about Java generics to solve problems-implement your own generic classes and methods.

-**App programmer** won’t write lots of generic code. Only code that traditionally involved **lots of casts** from very general types will **benefit** from using type parameters.

# 8.2 Defining a simple generic class

-**Generic class** have one or more **type variables**.

-Generic Pair class:



+A generic class can have **more than 1 type variable**:

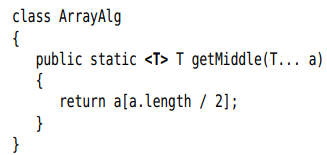


+The **type variables** are used throughout the **class definition** to specify **method return types** and the types of **fields** and **local variables**. 

+**Instantiate** the generic type by substituting types for the type variables: 

-**Note**: Use **uppercase letters** for type variables: **E** for element type of collection, **K** and **V** for key and value types of a table, **T** for any type at all.

# 8.3 Generic Methods



+Type variable <T> is between modifier and return type

-You can **define** **generic methods** both inside **ordinary classes** and **generic classes**.

-You can **omit** the **<T>** type parameter from **method call**





# 8.4 Bounds for Type Variables

-**Restrict** T to a class that **implements the interface**->give a **bound** for the type variable T: 

+T should be a **subtype** of the **bounding type**. Both T and the bounding type can be a **class** or an **interface**.

-A type variable or wildcard can have **multiple bounds**:



+The bounding types are separated by **&**.

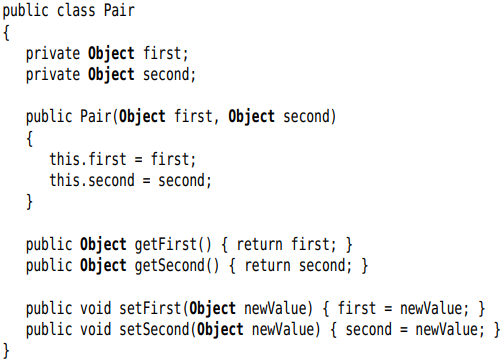
-You can have as many interfaces supertypes as you like.

# 8.5 Generic Code and the Virtual Machine

-The VM doesn’t have objects of generic types – all objects belong to ordinary classes.

## 8.5.1 Type Erasure

-When you define a generic type, a corresponding **raw** type is automatically provided. The name of the raw type is the name of generic type. The type variables are **erased** and replaced by their bounding types



+T is an unbounded type variable, it’s replaced by Object. The result is an ordinary class.

-**Note**: class Interval<T extends Serializable & Comparable>: raw type replaces T with Serializable, and the compiler inserts casts to Comparable when necessary. You should put tagging interface (interface without method) at the end of the bounds list.

## 8.5.2 Translating Generic Expressions

-When program call a generic method, the compiler inserts casts automatically when the return type has been erased: a call to the raw method + A cast of the returned Object to the type T.

-Casts are also inserted when accessing a generic field.

## 8.5.3 Translating Generic Methods

-Summary:

+There are no generics in the virtual machine, only ordinary classes and methods.

+All type parameters are replaced by their bounds.

+**Bridge methods** are synthesized to preserve polymorphism.

+Casts are inserted as necessary to preserve type safety.

## 8.5.4 Calling Legacy Code



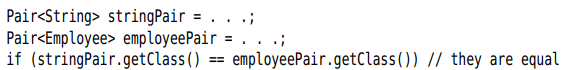
# 8.6 Restrictions and Limitations

## 8.6.1 Type Parameters cannot be instantiated with Primitive Types

-Because of type erasure.

## 8.6.2 Runtime Type Inquiry only works with raw Types

-Objects in VM always have a specific nongeneric type. So all type inquiries yield only the raw type:



## 8.6.3 You cannot create Arrays of Parameterized Types

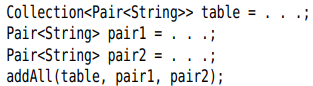
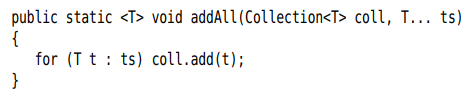


-**Note**: You can declare arrays of wildcard types and cast them: 

-If you need to collect parameterized type objects, use ArrayList<Pair<String>>

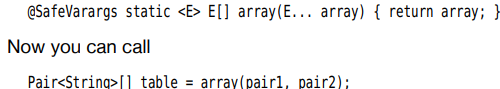
## 8.6.4 Varargs Warnings

-Passing instances of a generic type to a method with a variable number of arguments:



+VM must make an array of Pair<String>->get a warning. So add **@SuppressWarnings(“unchecked”)** to the method containing adAll() or add addAll() with **@SafeVarargs** (static, final or private).

-**Note**: Use **@SafeVarargs** to defeat the restriction against generic array creation:



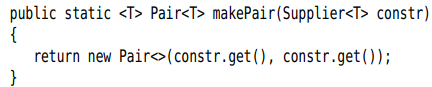
## 8.6.5 You cannot instantiate Type Variables

-Cannot use type variable in new T(…)

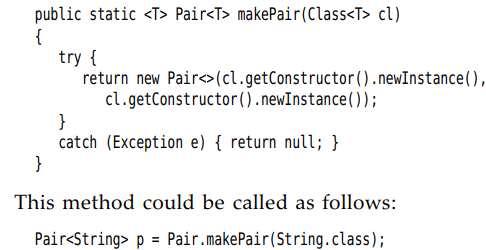


-Make the **caller** provide a **constructor** expression:





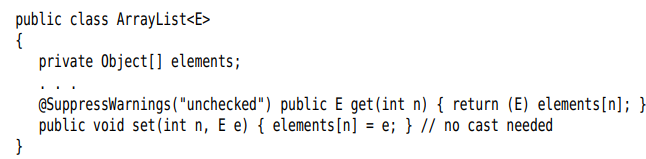
-A traditional workaround is to construct generic objects through reflection. You must design an API



+Class class is generic. String.class is an instance of Class<String>

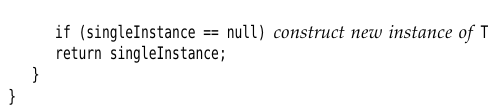
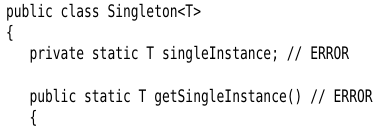
## 8.6.6 You cannot Construct a Generic Array

-If the array is only used as a private instance field of a class, you can declare the element type of the array to be the erased type and use casts:



## 8.6.7 Type Variables are not valid in static contexts of Generic Classes

-Can’t reference type variables in static fields or methods.

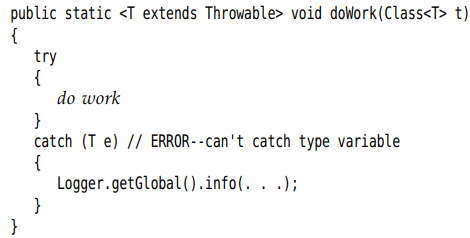


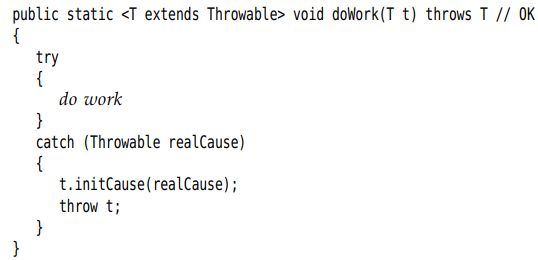
## 8.6.8 You cannot throw or catch instances of a Generic Class

-You cant throw or catch objects of a **generic class**:



-Can’t use type variable in catch:



-You can use type variables in exception specifications  


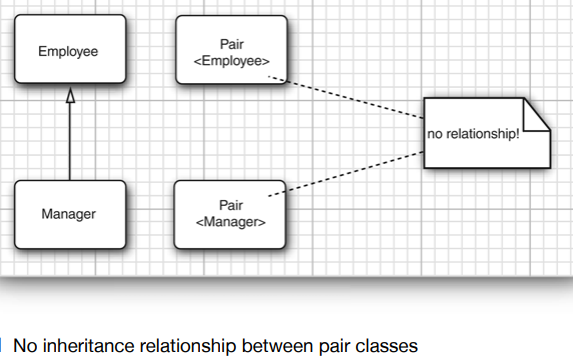
## 8.6.9 You can defeat Checked Exception Checking

## 8.6.10 Beware of Clashes after Erasure

# 8.7 Inheritance Rules for Generic Types

-There is no relationship between Pair<S> and Pair<T>, no matter how S and T are related.

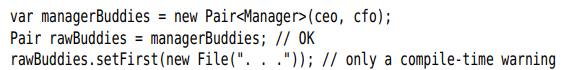




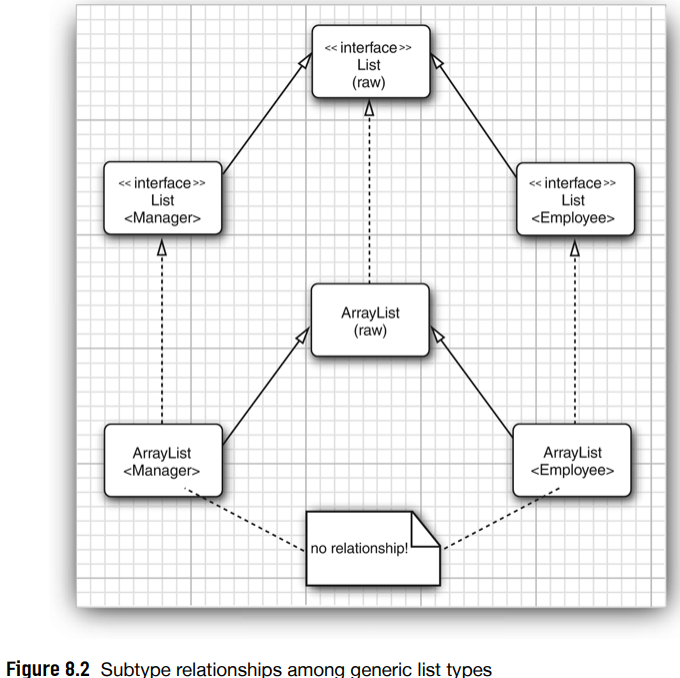
+**Note**: This is an important difference between generic types and Java arrays.

 but when you store a Employee into emoloyeeBuddies[0]-> throws ArrayStoreException

-You can convert a **parameterized type** to a **raw type**. Example: Pair<Employee> is a subtype of the raw type Pair -> Interfacing with legacy code.



-Generic classes can extend or implemented other generic classes.



+ArrayList<T> implement List<T> -> ArrayList<Manager> can be converted to List<Manager>, but ArrayList<Manager> isn’t an ArrayList<Employee> or List<Employee>

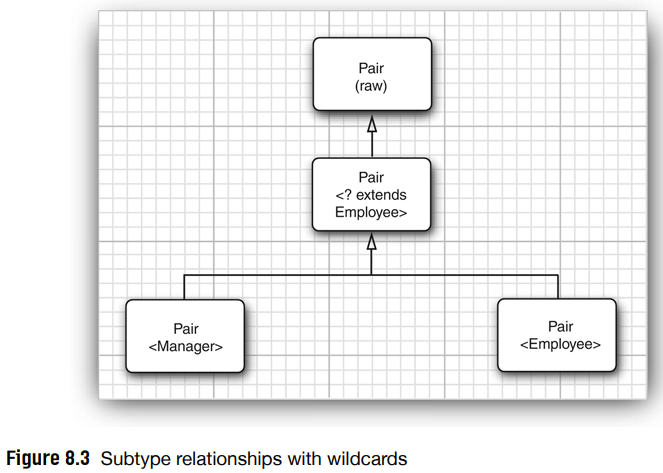
# 8.8 Wildcard Types

## 8.8.1 The Wildcard Concept

-In a **wildcard type**, a **type parameter** is allowed to **vary**:

 denotes any generic Pair type whose type parameter is a subclass of Employee.

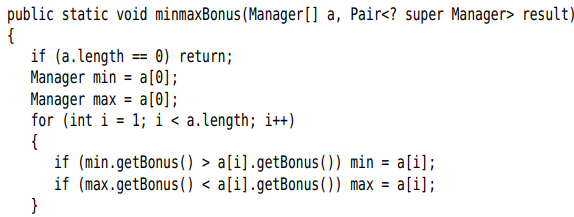


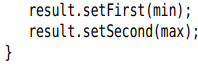


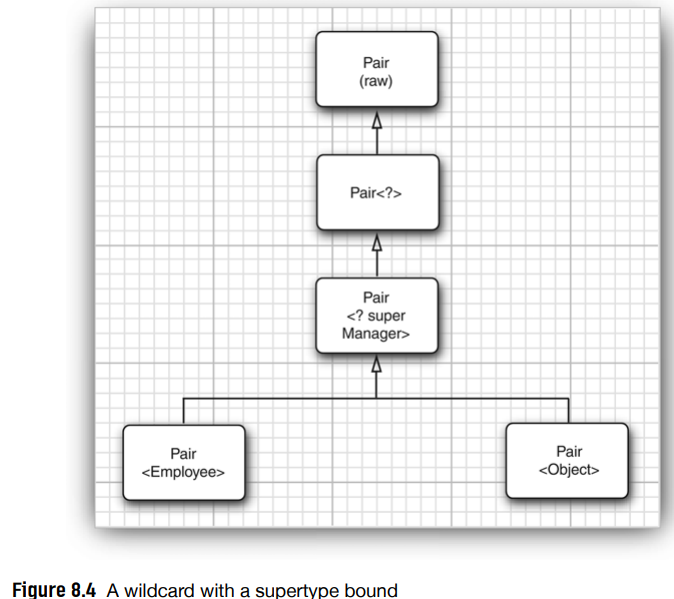
## 8.8.2 Supertype Bounds for Wildcards

-Supertype bound: . This wildcard is restricted to all **supertype** of Manager.

-Example:







-**Note**:Wildcard with **suptertype** bounds let you **write** to a generic object, while wildcard with **subtype** bounds let you **read** from a generic object.

-Use Comparable with generic:



compareTo(): 

-**Note**: Supertype bounds can be an argument type of a **functional interface**:





## 8.8.3 Unbounded Wildcards

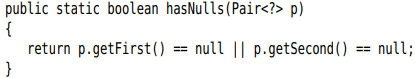
-Pair<?> no bounds:



+getFirst return only Object. setFirst can never be called

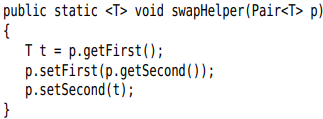
Note: You can call setFirst(null)

-It’s useful for very simple operation: test whether a pair contains null:



## 8.8.4 Wildcard Capture



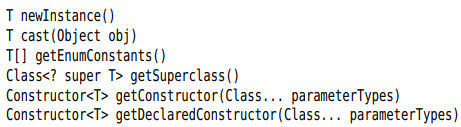


+swapHelper is a generic method, swap is not-it has a fixed parameter of type Pair<?>

+The parameter T of swapHelper method **captures the wildcard**.

# 8.9 Reflection and Generics

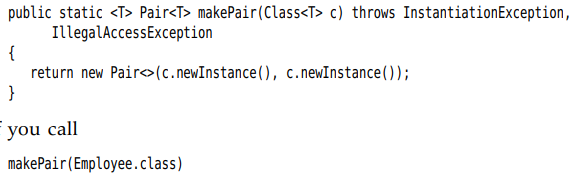
## 8.9.1 The Generic Class Class

-Class class is generic. String.class is the sole object of class Class<String>  


-API: p468

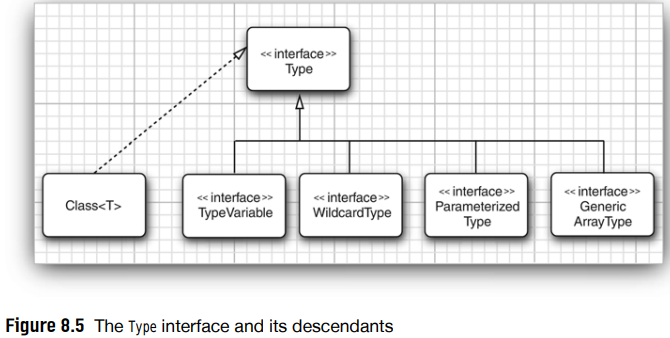
## 8.9.2 Using Class<T> Parameters for Type Matching

-Match the type variable of a Class<T> parameter in a generic method:



+Employee.class is an object of type Class<Employee>. The type parameter T of the makePair method matches Employee.

## 8.9.3 Generic Type Information in the Virtual Machine





+TypeVariable = T extends Comparable<? Super T>

+WildcardType = ? super T

+ParameterizedType = Comparable<? Super T>

+GenericArrayType = T[]

## 8.9.4 Type Literals

-Make difference actions for ArrayList<Integer> and ArrayList<String>

**-See 8.9 later**